第 20 卷 第 2 期 华侨大学学报(自然科学版) Vol. 20 No. 2
1999 年 4 月 Journal of Huaqiao University (Natural Science) Apr. 1999

基于点光源的多边形纹理处理技术*

张全伙 陈飞舟

(华侨大学计算机科学系, 泉州 362011)

摘要 在 Pentium 166, 32M RAM, Trident 9685 2M 显存的机器上和在 Microsoft Direct X5.0 DirectDraw 的支撑环境下, 采用 Borland C++ Builder 对 3D 引擎的低层次进行探讨. 文章假定在点光源照射下, 通过对多边形的三角形划分, 用插值法计算象素亮度, 实现三角形纹理的着色, 给出了三角形纹理处理的 Gouraud 模型的实现实例.

关键词 点光源, 纹理, 三角形划分, Gouraud 模型, 3D 引擎

分类号 TP 391.41

随着计算机芯片速度的不断提高, 原来用于大型机的真实感图形生成技术, 已逐步开始在微机上实现. 在具体实现上有两个不同的发展方向: 其一是, 应用计算机图形学的有关研究成果, 通过大量的计算来实现物体的细节部分, 达到生成极富真实感的图像. 这一类软件的典型代表是动画制作系统, 如 3DS, 3DMAX 等. 另一方向, 是在保证一定程度真实感的前提下, 通过简化和优化光照模型, 损失部分细节, 尽量减少计算量, 以达到生成图形的实时性. 这一类软件主要有虚拟环境构造[1]、飞行模拟器、3D 游戏等. 其中 3D 游戏最为常见, 最关键部分是 3D 引擎.

1 多边形的三角形划分

物体在 3D 引擎中一般以多面体表示, 它的每一个面是 1 个多边形. 考虑到对多边形进行纹理着色比较麻烦, 而三角形是最简单的凸多边形. 因此, 我们把多边形分解成一系列三角形, 然后对三角形进行纹理着色. 假设多边形的顶点数为 n, 并按顺时针方向排列, 划分后的三角形放在数组 Tri[$n-2$][3] 中. 把多边形划分为三角形有多种方法, 我们这里只讨论条形划分和扇形划分.

1.1 条形划分

将多边形从左到右按顺时针方向划分三角形. 它以 3 个三角形为 1 个周期, 在一个周期内 3 个三角形的顶点依次为$(i, i+1, n-i)$, $(i+1, i+2, n-i)$, $(n-i, i+2, n-i-1)$, 其中 i 为周期起始三角形的第 1 个顶点序号. 划分的终止条件是三角形第 2 点的顶点序号大于等于第 3 点的顶点序号减 1. 如图 1 所示的多边形, 可划分为(1, 2, 8), (2, 3, 8), (8, 3, 7), (3, 4, 7),

* 本文 1998-09-11 收到; 福建省自然科学基金资助项目

(4, 5, 7), (7, 5, 6) 等 6 个顺时针方向编号的三角形. 算法可描述如下

```
i= 1;
k= n;
t= 0;
While( 1){
Tri[t][0]= i;      Tri[t][1]= i+ 1;
Tri[t][2]= k;
if ( (i+ 1) > = (k- 1) ) break;
i+ + ;   t+ + ;
Tri[t][0]= i;      Tri[t][1]= i+ 1;      Tri[t][2]= k;
if (i+ 1) > = (k- 1) ) break;
k+ + ;   t+ + ;
Tri[t][0]= k;      Tri[t][1]= i;      Tri[t][2]= k- 1;
if ( (i > = (k- 2) ) break;
i- - ;   t+ + ;
}
```

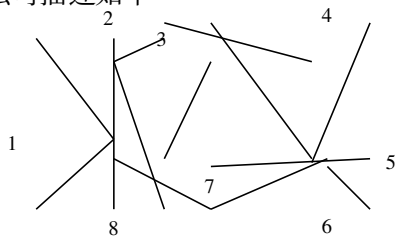


图1 条形划分

1.2 扇形划分

多边形的扇形划分, 以 1 个顶点为基点, 其它顶点按编号顺序和基点组成三角形. 如图 2 所示的多边形, 可划分为(1, 2, 3), (3, 4, 1), (1, 4, 5), (5, 6, 1), (1, 6, 7), (7, 8, 1), (1, 8, 9), (9, 10, 1) 等 8 个顺时针方向的三角形. 从图 2 中可以看出, 划分的三角形以 2 个三角形为 1 个周期, 下面所列为其实现算法.

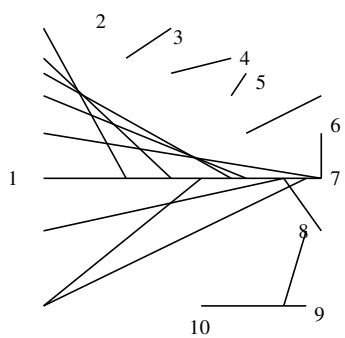


图2 扇形划分

```
t= 0;
i= 2;
While( 1){
Tri[t][0]= 1;      Tri[t][1]= i;      Tri[t][2]= i+ 1;
if ( i > = (n- 1) ) break;
t+ + ;   i+ + ;
Tri[t][0]= i;      Tri[t][1]= i+ 1;      Tri[t][2]= 1;
if ( i > = (n- 1) ) break;
t+ + ;   i+ + ;
{
```

多边形的条形划分和扇形划分对于复杂的凹多边形不一定适用, 但是在 3D 擎中一般采用简化的物体模型. 该物体表面通常为凸多边形, 并且目前的 3D 加速卡均支持这两种多边形划分方法, 使得这两种方法在 3D 引擎中仍得到普遍使用. 对于任意多边形的三角形划分可参阅文献[2].

2　顶点亮度计算[3]

物体表面多边形被划分为三角形后, 就要计算出各顶点的亮度值, 以表现物体的明暗效果. 在 3D 引擎中, 为了提高速度, 普遍采用 Gouraud 模型. 该模型只考虑物体反射光中的泛光和漫反射, 忽略了镜面反射部分. 为了更真实地表现物体的明暗, 可使用包含镜面反射的 Phong 模型[4]. 该模型对物体表面上的每一个点均计算出亮度值, 这在目前的主流机器上无法达到实时效果, 为此我们也只考虑泛光和漫反射.

(1) 泛光, 又称背景光或环境光. 是从环境中周围物体散射到物体表面再反射出来的光, 沿各个方向的亮度相同. 泛光的作用在于物体没有光源的情况下具有一定的亮度. 泛光的光强度为

$$I = I_a K_a,$$

其中 I_a 是环境中的泛光亮度值, K_a 是泛光的漫反射常数 $(0 \le K_a \le 1)$.

(2) 漫反射. 是光线穿过物体表面并被吸收, 然后又重新发射出来的光. 它沿各个方向都作相同的散射, 从各个视角看这种表面都有相同的亮度, 如图 3 所示. 漫反射的光强为

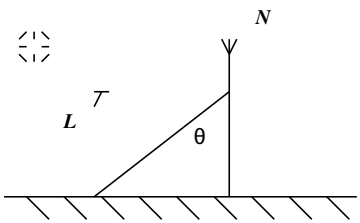


图3　漫反射示意图

$$I = K_d \quad I_l \quad \cos\theta \qquad 0 \le \theta \le \pi/2,$$

其中 I_l 为点光源光强, K_d 为漫反射常数 $(0 \le K_d \le 1)$, 由物体表面材料所决定. 设 $\boldsymbol{L}$ 是指向点光源的单位向量, $\boldsymbol{N}$ 是单位法向量, 于是有 $\cos\theta = \boldsymbol{N} \cdot \boldsymbol{L}$. 取 $K_a = K_d$, 忽略点光源与物体表面距离, 以及透视点与物体表面距离对光强的影响. 因此, 计算时的实际光强为

$$I = K_d \quad (I_a + I_l \quad (\boldsymbol{N} \quad \boldsymbol{L})),$$

对于多个点光源的情况, 则为

$$I = K_d \quad (I_a + \sum_{i=1}^{n} (I_{l_i} \quad (\boldsymbol{N}_i \quad \boldsymbol{L}_i))).$$

3　三角形的纹理处理[5]

所谓纹理就是物体的表面细节. 我们只考虑颜色纹理(又称平面纹理), 它是通过颜色或明暗度变化来表现物体的表面细节. 在 3D 引擎中, 一般用离散的方形阵列(数组)来表示, 阵列大小已由早期的 64×64 发展到最新的 512×512. 考虑到阵列太大占用空间也大, 为了达到容量和效果的平衡, 我们采用 256×256 大小的纹理空间.

Gouraud 模型的明暗处理方法是, 首先精确计算出三角形各个顶点的亮度值, 然后通过顶点亮度的线性插值得到三角形边界及内部各点的亮度. 在具体实现上, 可以和采用线性插值的扫描算法结合起来, 得到三角形中各点在纹理阵列中对应位置. 为此, 我们为线性插值算法定义了两个宏(Macro), 以简化程序设计.

```
# define LinerInit(Master, FirstValue, LastValue, Step) \
double Master; \
double Delta# # Master; \
\
```

```
Delta## Master= (double)((LastValue)-(FirstValue))/(double)Step; \
Master= (double) (FirstValue);
# define LinerNext(Master)Master += Delta## Master;
```

其中宏 LinerInit 用于初始化插值变量, Master 是插值变量, FirstValue 是初值, LastValue 是终值, Step 是插值变量从初值到终值需要的增量次数. 宏 LinerNext 用于插值变量取下一个增量结果.

下面我们采用扫描线算法来对三角形进行纹理着色. 假设显示缓冲区用 Color Screen[600][800]来表示, 纹理用 Color Texture[256][256]表示, 顶点的数据结构为

```
Struct Vertex{
double X, Y;          //顶点的位置
double Gray;          //顶点的亮度
double tu, tv;        //顶点的纹理阵列的位置
}
```

假设三角形的 3 个顶点为 P_0, P_1, P_2, 并且有 $P_0 \cdot Y \leq P_1 \cdot Y \leq P_2 \cdot Y$. 此外, 过 P_1 引水平线交于 P_0P_2 上点 P_3. 这样, 三角形 $P_0P_1P_2$ 被分割为上下两条底线水平的三角形 $P_0P_1P_3$ 和 $P_3P_1P_2$, 如图4 所示. 我们首先考虑三角形 $P_0P_1P_3$ 的纹理着色. 算法描述如下: (1) $Y = P_0 \cdot Y$; (2) 沿 P_0P_2 直线插值得插值变量 $X_0, Gray_0, tu_0, tv_0$; (3) 沿 P_0P_1 直线插值得插值变量 $X_1, Gray_1, tu_1, tv_1$; (4) 着色完成否? 是则退出, 否则继续; (5) 沿 X_0X_1 水平线插值得插值变量 $X, Gray, tu, tv$; (6) X_0X_1 行着色完成否? 是转(9); 否则继续; (7) 计算屏幕上对应象素点的颜色值: Screen[Y][X] = (Color) $Gray^*$ Texture[tv][tu]; (8) 更新插值变量 $X, Gray, tu, tv$; 转(6); (9) $Y = Y + 1$; (10) 更新插值变量 $X_0, Gray_0, tu_0, tv_0, X_1, Gray_1, tu_1, tv_1$; 转(4). 其相应程序段为

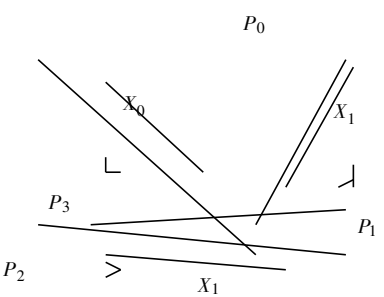


图4 三角形的着色顺序

```
double Y= P0 · Y;
//沿 P0P2 直线插值得插值变量 X0, Gray0, tu0, tv0
double P0P2Y= P2 · Y- P0 · Y;
LinerInit(X0, P0 · X, P2 · X, P0P2Y);
LinerInit(Gray0, P0 · Gray, P2 · Gray, P0P2Y);
LinerInit(tu0, P0, tu, P2 · tu, P0P2Y);
LinerInit(tv0, P0 · tv, P2 · tv, P0P2Y);
//沿 P0P1 直线插值得插值变量 X1, Gray1, tu1, tv1
double P0P1Y= P1 · Y- P0 · Y;
LinerInit(X1, P0 · X, P1 · X, P0P1Y);
LinerInit(Gray1, P0 · Gray, P1 · Gray, P0P1Y);
LinerInit(tu1, P0, tu, P1 · tu, P0P1Y);
LinerInit(tv1, P0 · tv, P1 · tv, P0P1Y);
```

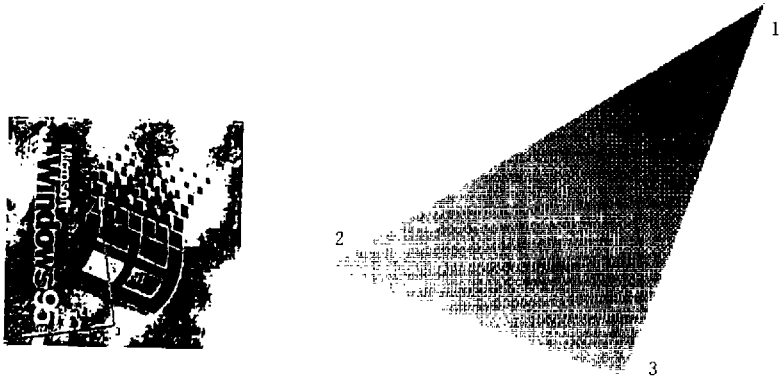
```
While(true){
if ( Y > = P1 · Y ) break;// 如果着色结束则退出
//沿 X0X1 水平线插值得插值变量 X, Gray, tu, tv
double X0X1= X1- X0;
LinerInit(X, X0, X1, X1, X0X1);
LinerInit(Gray, Gray0, Gray1, X0X1);
LinerInit(tu, tu0, tu1, X0X1);
LinerInit(tv, tv0, tv1, X0X1);
While(true){
if ( X > = X1) break;// 如果行着色完成则转下一行
//计算屏幕上相应点的颜色值
Screen[ (int) Y] [ (int) X] =
(Color)(Gray* Texture[(int)tv] [(int)tu]);
//更新插值变量
LinerNext(X);
LinerNext(Gray);
LinerNext(tu);
LinerNext(tv);
}
//下一行
Y = Y + 1;
//更新插值变量
LinerNext(X0);
LinerNext(Gray0);
LinerNext(tu0);
LinerNext(tv0);
LinerNext(X1);
LinerNext(Gray1);
LinerNext(tu1);
LinerNext(tv1);
}
```

按照同样的方法, 我们可以得出下三角形 $P_3P_1P_2$ 的着色程序, 将这两部分程序合在一起便可实现三角形的纹理着色 .

4　效果图实例与说明

在点光源照射下, 采用 800×600×16 位色模式实现的三角形纹理着色的效果图 . 图 5 (a) 是纹理原图, 三角形是效果图在纹理原图中的映射三角形; 图 5(b) 是三角形着色效果图 . 从中我们可以看出, 整个三角形的明暗过渡自然, 效果逼真 .

我们讨论了3D引擎中底层多边形的三角形划分、顶点亮度计算、三角形纹理着色算法的3个重要方面,并给出具体的实现.尽管我们对Gourand模型进行了简化和优化,但在用C++



(a)纹理原图 (b)三角形着色效果

图5 纹理映射

实现三角形纹理着色的程序中,一般大小的三角形在Pentium 166机器上800×600×16位色模式的着色速度仅每秒32个,远远达不到实时效果.可见,要生成富有真实感的图形,又要做到实时性,对机器的容量和速度有多么高的要求.

参 考 文 献

1 汪成为,高 文,王行仁.灵境(虚拟现实)技术的理论、实现及应用.北京:清华大学出版社,1997.121～126
2 陈 真,陈 伟.计算机图形学.北京:北京航空航天大学出版社,1990.203～207,233～239
3 孙家广,杨长贵.计算机图形学(新版).北京:清华大学出版社,1994.481～485
4 罗振东,廖光裕.计算机图示原理和方法.上海:复旦大学出版社,1993.299～305
5 张全伙,骆炎民.微机动机及其应用.华侨大学学报(自然科学版),1997,18(2):83～86

Technology Based on Point Soure of light for Processing Textures on a Polygon

Zhang Quanhuo Chen Feizhou
(Dept. of Comput. Sci., Huaqiao Univ., 362011, Quanzhou)

Abstract On the machine with Pentium 166, 32M RAM、Trident 9685 2M as display memory and with Microsoft Direct X5.0 Directdraw as support environment, the authors probe into a 3D engine at low level by adopting Borland C++ Builder. With the irrdiation of point sourse of light, a polygon is assumed to be partited into triangles and the brightness of image elements is computed by interpolation method. As an example for realizing the triangular texture processing, the Gouraud model is given finally.

Keywords point source of light, texture, triangular partition, Gouraud model, 3D engine